

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

EV316935389

Programming Interface for a Computer Platform

Inventor(s):

**Kerem Karatal
Mike Sheldon
Marc Miller
Chris Guzak
Tim McKee**

ATTORNEY'S DOCKET NO. MS1-1792US

1 **TECHNICAL FIELD**

2 This invention relates to software and to development of such software.
3 More particularly, this invention relates to a programming interface that facilitates
4 use of a software platform by application programs and computer hardware.

5

6 **BRIEF DESCRIPTION OF ACCOMPANYING COMPACT DISCS**

7 Accompanying this specification is a set of three compact discs that stores a
8 Software Development Kit (SDK) for the Microsoft® Windows® Code-Named
9 "Longhorn" operating system. The SDK contains documentation for the
10 Microsoft® Windows® Code-Named "Longhorn" operating system. Duplicate
11 copies of each of these three compact discs also accompany this specification.

12 The first compact disc in the set of three compact discs (CD 1 of 3) includes a file folder named "lhsdk" that was created on October 22, 2003; it is 586 Mbytes in size, contains 9,692 sub-folders, and contains 44,292 sub-files. The second compact disc in the set of three compact discs (CD 2 of 3) includes a file folder named "ns" that was created on October 22, 2003; it is 605 Mbytes in size, contains 12,628 sub-folders, and contains 44,934 sub-files. The third compact disc in the set of three compact discs (CD 3 of 3) includes a file folder named "ns" that was created on October 22, 2003; it is 575 Mbytes in size, contains 9,881 sub-folders, and contains 43,630 sub-files. The files on each of these three compact discs can be executed on a Windows®-based computing device (e.g., IBM-PC, or equivalent) that executes a Windows®-brand operating system (e.g., Windows® NT, Windows® 98, Windows® 2000, Windows® XP, etc.). The files on each compact disc in this set of three compact discs are hereby incorporated by reference.

1 Each compact disc in the set of three compact discs itself is a CD-R, and
2 conforms to the ISO 9660 standard. The contents of each compact disc in the set
3 of three compact discs is in compliance with the American Standard Code for
4 Information Interchange (ASCII).

5

6 **BACKGROUND**

7 Very early on, computer software came to be categorized as “operating
8 system” software or “application” software. Broadly speaking, an application is
9 software meant to perform a specific task for the computer user such as solving a
10 mathematical equation or supporting word processing. The operating system is
11 the software that manages and controls the computer hardware. The goal of the
12 operating system is to make the computer resources available to the application
13 programmer while at the same time, hiding the complexity necessary to actually
14 control the hardware.

15 The operating system makes the resources available via functions that are
16 collectively known as the Application Program Interface or API. The term API is
17 also used in reference to a single one of these functions. The functions are often
18 grouped in terms of what resource or service they provide to the application
19 programmer. Application software requests resources by calling individual API
20 functions. API functions also serve as the means by which messages and
21 information provided by the operating system are relayed back to the application
22 software.

23 In addition to changes in hardware, another factor driving the evolution of
24 operating system software has been the desire to simplify and speed application
25 software development. Application software development can be a daunting task,

1 sometimes requiring years of developer time to create a sophisticated program
2 with millions of lines of code. For a popular operating system such as various
3 versions of the Microsoft Windows® operating system, application software
4 developers write thousands of different applications each year that utilize the
5 operating system. A coherent and usable operating system base is required to
6 support so many diverse application developers.

7 Often, development of application software can be made simpler by making
8 the operating system more complex. That is, if a function may be useful to several
9 different application programs, it may be better to write it once for inclusion in the
10 operating system, than requiring dozens of software developers to write it dozens
11 of times for inclusion in dozens of different applications. In this manner, if the
12 operating system supports a wide range of common functionality required by a
13 number of applications, significant savings in applications software development
14 costs and time can be achieved.

15 Regardless of where the line between operating system and application
16 software is drawn, it is clear that for a useful operating system, the API between
17 the operating system and the computer hardware and application software is as
18 important as efficient internal operation of the operating system itself.

19 Over the past few years, the universal adoption of the Internet, and
20 networking technology in general, has changed the landscape for computer
21 software developers. Traditionally, software developers focused on single-site
22 software applications for standalone desktop computers, or LAN-based computers
23 that were connected to a limited number of other computers via a local area
24 network (LAN). Such software applications were typically referred to as “shrink-
25 wrapped” products because the software was marketed and sold in a shrink-

1 wrapped package. The applications utilized well-defined APIs to access the
2 underlying operating system of the computer.

3 As the Internet evolved and gained widespread acceptance, the industry
4 began to recognize the power of hosting applications at various sites on the World
5 Wide Web (or simply the “Web”). In the networked world, clients from anywhere
6 could submit requests to server-based applications hosted at diverse locations and
7 receive responses back in fractions of a second. These Web applications, however,
8 were typically developed using the same operating system platform that was
9 originally developed for standalone computing machines or locally networked
10 computers. Unfortunately, in some instances, these applications do not adequately
11 transfer to the distributed computing regime. The underlying platform was simply
12 not constructed with the idea of supporting limitless numbers of interconnected
13 computers.

14 To accommodate the shift to the distributed computing environment being
15 ushered in by the Internet, Microsoft Corporation developed a network software
16 platform known as the “.NET” Framework (read as “Dot Net”). Microsoft® .NET
17 is software for connecting people, information, systems, and devices. The
18 platform allows developers to create Web services that will execute over the
19 Internet. This dynamic shift was accompanied by a set of API functions for
20 Microsoft's .NET™ Framework.

21 As use of the .NET™ Framework has become increasingly common, ways
22 to increase the efficiency and/or performance of the platform have been identified.
23 The inventors have developed a unique set of programming interface functions to
24 allow for such increased efficiency and/or performance.

25

1 **SUMMARY**

2 A programming interface for a computer platform is described herein.

3 In accordance with certain aspects, the programming interface can include
4 one or more of the following groups of services: a first group of services related
5 to re-usable user interface controls, a second group of services related to user
6 interface dialogs and user interface wizards, a third group of services related to
7 extending the user interface functionality, and a fourth group of services related to
8 extending functionality of a desktop of the user interface.

9

10 **BRIEF DESCRIPTION OF THE DRAWINGS**

11 The same numbers are used throughout the drawings to reference like
12 features.

13 Fig. 1 illustrates a network architecture in which clients access Web
14 services over the Internet using conventional protocols.

15 Fig. 2 is a block diagram of a software architecture for a network platform,
16 which includes an application program interface (API).

17 Fig. 3 is a block diagram of unique namespaces supported by the API, as
18 well as function classes of the various API functions.

19 Fig. 4 is a block diagram of an exemplary computer that may execute all or
20 part of the software architecture.

21 Figs. 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, and 16 illustrate various example
22 implementations of a programming interface.

DETAILED DESCRIPTION

This disclosure addresses a programming interface, such as an application program interface (API), for a network platform upon which developers can build Web applications and services. More particularly, an exemplary API is described for operating systems that make use of a network platform, such as the .NET™ Framework created by Microsoft Corporation. The .NET™ Framework is a software platform for Web services and Web applications implemented in the distributed computing environment. It represents the next generation of Internet computing, using open communication standards to communicate among loosely coupled Web services that are collaborating to perform a particular task.

In the described implementation, the network platform utilizes XML (extensible markup language), an open standard for describing data. XML is managed by the World Wide Web Consortium (W3C). XML is used for defining data elements on a Web page and business-to-business documents. XML uses a similar tag structure as HTML; however, whereas HTML defines how elements are displayed, XML defines what those elements contain. HTML uses predefined tags, but XML allows tags to be defined by the developer of the page. Thus, virtually any data items can be identified, allowing Web pages to function like database records. Through the use of XML and other open protocols, such as Simple Object Access Protocol (SOAP), the network platform allows integration of a wide range of services that can be tailored to the needs of the user. Although the embodiments described herein are described in conjunction with XML and other open standards, such are not required for the operation of the claimed invention. Other equally viable technologies will suffice to implement the inventions described herein.

1 As used herein, the phrase application program interface or API includes
2 traditional interfaces that employ method or function calls, as well as remote calls
3 (e.g., a proxy, stub relationship) and SOAP/XML invocations.

5 EXEMPLARY NETWORK ENVIRONMENT

6 Fig. 1 shows a network environment 100 in which a network platform, such
7 as the .NET™ Framework, may be implemented. The network environment 100
8 includes representative Web services 102(1), ..., 102(N), which provide services
9 that can be accessed over a network 104 (e.g., Internet). The Web services,
10 referenced generally as number 102, are programmable application components
11 that are reusable and interact programmatically over the network 104, typically
12 through industry standard Web protocols, such as XML, SOAP, WAP (wireless
13 application protocol), HTTP (hypertext transport protocol), and SMTP (simple
14 mail transfer protocol) although other means of interacting with the Web services
15 over the network may also be used, such as Remote Procedure Call (RPC) or
16 object broker type technology. A Web service can be self-describing and is often
17 defined in terms of formats and ordering of messages.

18 Web services 102 are accessible directly by other services (as represented
19 by communication link 106) or a software application, such as Web application
20 110 (as represented by communication links 112 and 114). Each Web service 102
21 is illustrated as including one or more servers that execute software to handle
22 requests for particular services. Such services often maintain databases that store
23 information to be served back to requesters. Web services may be configured to
24 perform any one of a variety of different services. Examples of Web services
25 include login verification, notification, database storage, stock quoting, location

1 directories, mapping, music, electronic wallet, calendar/scheduler, telephone
2 listings, news and information, games, ticketing, and so on. The Web services can
3 be combined with each other and with other applications to build intelligent
4 interactive experiences.

5 The network environment 100 also includes representative client devices
6 120(1), 120(2), 120(3), 120(4), ..., 120(M) that utilize the Web services 102 (as
7 represented by communication link 122) and/or the Web application 110 (as
8 represented by communication links 124, 126, and 128). The clients may
9 communicate with one another using standard protocols as well, as represented by
10 an exemplary XML link 130 between clients 120(3) and 120(4).

11 The client devices, referenced generally as number 120, can be
12 implemented many different ways. Examples of possible client implementations
13 include, without limitation, portable computers, stationary computers, tablet PCs,
14 televisions/set-top boxes, wireless communication devices, personal digital
15 assistants, gaming consoles, printers, photocopiers, and other smart devices.

16 The Web application 110 is an application designed to run on the network
17 platform and may utilize the Web services 102 when handling and servicing
18 requests from clients 120. The Web application 110 is composed of one or more
19 software applications 130 that run atop a programming framework 132, which are
20 executing on one or more servers 134 or other computer systems. Note that a
21 portion of Web application 110 may actually reside on one or more of clients 120.
22 Alternatively, Web application 110 may coordinate with other software on clients
23 120 to actually accomplish its tasks.

24 The programming framework 132 is the structure that supports the
25 applications and services developed by application developers. It permits multi-

language development and seamless integration by supporting multiple languages. It supports open protocols, such as SOAP, and encapsulates the underlying operating system and object model services. The framework provides a robust and secure execution environment for the multiple programming languages and offers secure, integrated class libraries.

The framework 132 is a multi-tiered architecture that includes an application program interface (API) layer 142, a common language runtime (CLR) layer 144, and an operating system/services layer 146. This layered architecture allows updates and modifications to various layers without impacting other portions of the framework. A common language specification (CLS) 140 allows designers of various languages to write code that is able to access underlying library functionality. The specification 140 functions as a contract between language designers and library designers that can be used to promote language interoperability. By adhering to the CLS, libraries written in one language can be directly accessible to code modules written in other languages to achieve seamless integration between code modules written in one language and code modules written in another language. One exemplary detailed implementation of a CLS is described in an ECMA standard created by participants in ECMA TC39/TG3.

The reader is directed to the ECMA web site at www.ecma.ch.

The API layer 142 presents groups of functions that the applications 130 can call to access the resources and services provided by layer 146. By exposing the API functions for a network platform, application developers can create Web applications for distributed computing systems that make full use of the network resources and other Web services, without needing to understand the complex interworkings of how those network resources actually operate or are made

1 available. Moreover, the Web applications can be written in any number of
2 programming languages, and translated into an intermediate language supported
3 by the common language runtime 144 and included as part of the common
4 language specification 140. In this way, the API layer 142 can provide methods
5 for a wide and diverse variety of applications.

6 Additionally, the framework 132 can be configured to support API calls
7 placed by remote applications executing remotely from the servers 134 that host
8 the framework. Representative applications 148(1) and 148(2) residing on clients
9 120(3) and 120(M), respectively, can use the API functions by making calls
10 directly, or indirectly, to the API layer 142 over the network 104.

11 The framework may also be implemented at the clients. Client 120(3)
12 represents the situation where a framework 150 is implemented at the client. This
13 framework may be identical to server-based framework 132, or modified for client
14 purposes. Alternatively, the client-based framework may be condensed in the
15 event that the client is a limited or dedicated function device, such as a cellular
16 phone, personal digital assistant, handheld computer, or other
17 communication/computing device.

18

19 DEVELOPERS' PROGRAMMING FRAMEWORK

20 Fig. 2 shows the programming framework 132 in more detail. The
21 common language specification (CLS) layer 140 supports applications written in a
22 variety of languages 130(1), 130(2), 130(3), 130(4), ..., 130(K). Such application
23 languages include Visual Basic, C++, C#, COBOL, Jscript, Perl, Eiffel, Python,
24 and so on. The common language specification 140 specifies a subset of features
25 or rules about features that, if followed, allow the various languages to

1 communicate. For example, some languages do not support a given type (e.g., an
2 “int*” type) that might otherwise be supported by the common language runtime
3 144. In this case, the common language specification 140 does not include the
4 type. On the other hand, types that are supported by all or most languages (e.g.,
5 the “int[]” type) is included in common language specification 140 so library
6 developers are free to use it and are assured that the languages can handle it. This
7 ability to communicate results in seamless integration between code modules
8 written in one language and code modules written in another language. Since
9 different languages are particularly well suited to particular tasks, the seamless
10 integration between languages allows a developer to select a particular language
11 for a particular code module with the ability to use that code module with modules
12 written in different languages. The common language runtime 144 allow seamless
13 multi-language development, with cross language inheritance, and provide a
14 robust and secure execution environment for the multiple programming languages.
15 For more information on the common language specification 140 and the common
16 language runtime 144, the reader is directed to co-pending applications entitled
17 “Method and System for Compiling Multiple Languages”, filed 6/21/2000 (serial
18 number 09/598,105) and “Unified Data Type System and Method” filed 7/10/2000
19 (serial number 09/613,289), which are incorporated by reference.

20 The framework 132 encapsulates the operating system 146(1) (e.g.,
21 Windows®-brand operating systems) and object model services 146(2) (e.g.,
22 Component Object Model (COM) or Distributed COM). The operating system
23 146(1) provides conventional functions, such as file management, notification,
24 event handling, user interfaces (e.g., windowing, menus, dialogs, etc.), security,
25 authentication, verification, processes and threads, memory management, and so

1 on. The object model services 146(2) provide interfacing with other objects to
2 perform various tasks. Calls made to the API layer 142 are handed to the common
3 language runtime layer 144 for local execution by the operating system 146(1)
4 and/or object model services 146(2).

5 The API 142 groups API functions into multiple namespaces. Namespaces
6 essentially define a collection of classes, interfaces, delegates, enumerations, and
7 structures, which are collectively called “types”, that provide a specific set of
8 related functionality. A class represents managed heap allocated data that has
9 reference assignment semantics. A delegate is an object oriented function pointer.
10 An enumeration is a special kind of value type that represents named constants. A
11 structure represents static allocated data that has value assignment semantics. An
12 interface defines a contract that other types can implement.

13 By using namespaces, a designer can organize a set of types into a
14 hierarchical namespace. The designer is able to create multiple groups from the
15 set of types, with each group containing at least one type that exposes logically
16 related functionality. In the exemplary implementation, the API 142 is organized
17 to include three root namespaces. It should be noted that although only three root
18 namespaces are illustrated in Fig. 2, additional root namespaces may also be
19 included in API 142. The three root namespaces illustrated in API 142 are: a first
20 namespace 200 for a presentation subsystem (which includes a namespace 202 for
21 a user interface shell), a second namespace 204 for web services, and a third
22 namespace 206 for a file system. Each group can then be assigned a name. For
23 instance, types in the presentation subsystem namespace 200 can be assigned the
24 name “Windows”, and types in the file system namespace 206 can be assigned
25 names “Storage”. The named groups can be organized under a single “global

1 “root” namespace for system level APIs, such as an overall System namespace. By
2 selecting and prefixing a top level identifier, the types in each group can be easily
3 referenced by a hierarchical name that includes the selected top level identifier
4 prefixed to the name of the group containing the type. For instance, types in the
5 file system namespace 206 can be referenced using the hierarchical name
6 “System.Storage”. In this way, the individual namespaces 200, 204, and 206
7 become major branches off of the System namespace and can carry a designation
8 where the individual namespaces are prefixed with a designator, such as a
9 “System.” prefix.

10 The presentation subsystem namespace 200 pertains to programming and
11 content development. It supplies types that allow for the generation of
12 applications, documents, media presentations and other content. For example,
13 presentation subsystem namespace 200 provides a programming model that allows
14 developers to obtain services from the operating system 146(1) and/or object
15 model services 146(2).

16 The shell namespace 202 pertains to user interface functionality. It supplies
17 types that allow developers to embed user interface functionality in their
18 applications, and further allows developers to extend the user interface
19 functionality.

20 The web services namespace 204 pertains to an infrastructure for enabling
21 creation of a wide variety of applications, e.g. applications as simple as a chat
22 application that operates between two peers on an intranet, and/or as complex as a
23 scalable Web service for millions of users. The described infrastructure is
24 advantageously highly variable in that one need only use those parts that are
25 appropriate to the complexity of a particular solution. The infrastructure provides

1 a foundation for building message-based applications of various scale and
2 complexity. The infrastructure or framework provides APIs for basic messaging,
3 secure messaging, reliable messaging and transacted messaging. In the
4 embodiment described below, the associated APIs have been factored into a
5 hierarchy of namespaces in a manner that has been carefully crafted to balance
6 utility, usability, extensibility and versionability.

7 The file system namespace 206 pertains to storage. It supplies types that
8 allow for information storage and retrieval.

9 In addition to the framework 132, programming tools 210 are provided to
10 assist the developer in building Web services and/or applications. One example of
11 the programming tools 210 is Visual Studio™, a multi-language suite of
12 programming tools offered by Microsoft Corporation.

14 ROOT API NAMESPACES

15 Fig. 3 shows the presentation subsystem namespace 200 in more detail. In
16 one embodiment, the namespaces are identified according to a hierarchical naming
17 convention in which strings of names are concatenated with periods. For instance,
18 the presentation subsystem namespace 200 is identified by the root name
19 “System.Windows”. Within the “System.Windows” namespace is another
20 namespace for the shell, identified as “System.Windows.Explorer”, which further
21 identifies another namespace for controls known as
22 “System.Windows.Explorer.Controls”. With this naming convention in mind, the
23 following provides a general overview of presentation subsystem namespace 200,
24 although other naming conventions could be used with equal effect.

1 The Shell namespace 202 ("System.Windows.Explorer"), includes classes
2 and APIs that support user interface functionality to allow end users to explore and
3 navigate to a set of endpoints, such as items in storage systems, files in FTP
4 locations or DAV (Distributed Authoring and Versioning) locations, control panel
5 applets in control panels, applications, and so forth. This navigation can involve
6 opening of one or more folders, as well as opening folders within folders. The
7 Shell namespace 202 allows developers to embed such functionality in their
8 applications, and further allows developers to extend this exploration and
9 navigation functionality. In versions of the Windows® operating system, this user
10 interface is typically referred to as the "Explorer".

11 In certain embodiments, the storage system that the user interface
12 functionality of shell namespace 202 allows users to explore and navigate through
13 is an active storage platform for organizing, searching for, and sharing all kinds of
14 information. This platform defines a rich data model, builds on top of a relational
15 storage engine, supports a flexible programming model, and provides a set of data
16 services for monitoring, managing, and manipulating data. The data can be file-
17 based or non-file data, and data is typically referred to as an "item". The file
18 system extends the functionality typically provided by file systems because it also
19 deals with items that are non-file data, such as personal contacts, event calendars,
20 and e-mail messages. One example of such a file system is the "WinFS" file
21 system.

22 A universal data store of the storage system that the user interface
23 functionality of shell namespace 202 allows users to explore and navigate through
24 implements a data model that supports the organization, searching, sharing,
25 synchronization, and security of data that resides in the store. The fundamental

1 unit of storage information in this data store is referred to as an item. The data
2 model provides a mechanism for declaring items and item extensions, for
3 establishing relationships between items, and for organizing items in folders and in
4 categories.

5 An item is a unit of storable information that, unlike a simple file, is an
6 object having a basic set of properties that are commonly supported across all
7 objects exposed to a user or application program by the storage system. Items also
8 have properties and relationships that are commonly supported across all item
9 types including features that allow new properties and relationships to be
10 introduced. This property and relationship data may also be referred to as
11 metadata associated with an item. As discussed in more detail below, the
12 metadata may be stored in accordance with an item decoration schema. This item
13 decoration schema may indicate an appropriate manner in which to present the
14 item to a user.

15 Items are the objects for common operations such as copy, delete, move,
16 open, print, backup, restore, replicate, and so forth. Items are the units that can be
17 stored and retrieved, and all forms of storable information manipulated by the
18 storage platform exist as items, properties of items, or relationships between items,
19 each of which is discussed in greater detail herein below. Items are intended to
20 represent real-world and readily-understandable units of data like Contacts,
21 People, Services, Locations, Documents (of all various sorts), and so on.

22 Items are stand-alone objects; thus, if an item is deleted, all of the
23 properties of the item are also deleted. Similarly, when retrieving an item, what is
24 received is the item and all of its properties contained in the metadata of the item.
25 Certain embodiments may enable one to request a subset of properties when

1 retrieving a specific item; however, the default for many such embodiments is to
2 provide the item with all of its immediate and inherited properties when retrieved.
3 Moreover, the properties of items can also be extended by adding new properties
4 to the existing properties of that item's type. These "extensions" are thereafter
5 bona fide properties of the item and subtypes of that item type may automatically
6 include the extension properties. The extensions may also be referred to as
7 metadata associated with a file.

8 Groups of items can be organized into special items called item Folders
9 (which are not to be confused with file folders). Unlike in most file systems,
10 however, an item can belong to more than one item Folder, such that when an item
11 is accessed in one item Folder and revised, this revised item can then be accessed
12 directly from another item Folder. In essence, although access to an item may
13 occur from different item Folders, what is actually being accessed is in fact the
14 very same item. However, an item Folder does not necessarily own all of its
15 member items, or may simply co-own items in conjunction with other item
16 Folders, such that the deletion of an item Folder does not necessarily result in the
17 deletion of the item.

18 Groups of items can also be organized into Categories. Categories are
19 conceptually different from item Folders in that, whereas item Folders may
20 comprise items that are not interrelated (i.e., without a common described
21 characteristic), each item in a Category has a common type, property, or value (a
22 "commonality") that is described for that Category, and it is this commonality that
23 forms the basis for its relationship to and among the other items in the Category.
24 Moreover, whereas an item's membership in a particular Folder is not compulsory
25 based on any particular aspect of that item, for certain embodiments all items

1 having a commonality categorically related to a Category might automatically
2 become a member of the Category at the hardware/software interface system level.
3 Conceptually, Categories can also be thought of as virtual item Folders whose
4 membership is based on the results of a specific query (such as in the context of a
5 database), and items that meet the conditions of this query (defined by the
6 commonalities of the Category) would thus comprise the Category's membership.

7 In contrast to files, folders, and directories, the items, item Folders, and
8 Categories of the present invention are not characteristically "physical" in nature
9 because they do not have conceptual equivalents of physical containers, and
10 therefore items may exist in more than one such location. The ability for items to
11 exist in more than one item Folder location as well as being organized into
12 Categories provides an enhanced and enriched degree of data manipulation and
13 storage structure capabilities at the hardware/software interface level, beyond that
14 currently available in the art.

15 Items may also contain relational information which allows relationships
16 between two or more items to be determined. Relationships are binary
17 relationships where one item is designated as source and the other item as target.
18 The source item and the target item are related by the relationship.

19 Items in the universal data store are presented to a user by a shell browser
20 that provides a user interface (using the classes and APIs of Shell namespace 202)
21 allowing a user to view and to interact with the hardware/software interface. Each
22 item in the universal data store is stored in accordance with a universal data
23 schema. This schema includes a mechanism for describing items called type
24 associations. Each type association has a basic representation in the shell; by
25

1 storing an item in accordance with a type association, the shell is able to display an
2 item according to at least a basic or default display view.

3 A type association is a property associated with an item; when placing data
4 into the universal data store one or more properties associated with the data must
5 be declared so as to determine what type of item it is. These properties may be
6 included as metadata associated with the data. The shell has a set of default type
7 associations which represent the most basic and minimal properties which must be
8 declared for an item.

9 Beyond property declarations, metadata associated with an item may
10 include data indicating how the shell should decorate an item's presentation.
11 Decorations, in this case can be thought of as "hints" as to how to represent the
12 item to a user. This metadata may be stored in accordance with an item decoration
13 schema. The item decoration schema defines the item decoration view that the
14 shell may utilize to present the item. For example, the item decoration data may
15 describe the most important declared properties for an item. These "high value"
16 properties may be the most desirable for presentation in the shell.

17 To present an item, the item decoration data may indicate a set of view
18 fields appropriate for the presentation of the item. View fields are projections of
19 declared properties, and common view fields may include "title," "author," "date
20 of creation" or "last edited." The shell includes a set of standard view fields and
21 independent software vendors (ISVs) may define view fields which are appropriate
22 for presentation of their data. When developing new item types, ISVs can either
23 map item properties they define to the shell's view fields or they can provide their
24 own view fields.

25

1 For example, particular item data may contain song data. The set of
2 declared properties may include properties such as song title, artist, date recorded,
3 album, song length, and other declarations appropriate for such a song item. The
4 item decoration data may indicate that view fields “Title,” “Artist” and “Album”
5 should be displayed to the user when presenting the item in the shell.

6 Item decorations can be any aspect of the display supported by the shell.
7 Some common other item decorations are, for example, data formatting, default
8 sort order, and default icon size. Additionally, the item decoration data may
9 describe common controls to use in displaying a given item. For example, a
10 Ratings field might use a ratings control that represent the rating as a series of
11 stars. The item decoration data may describe tasks or verbs appropriate for use
12 with an item. The terms “task” and “verb” describe some action to be undertaken
13 with regard to an item and such terms may be used interchangeably. For example,
14 “Edit” or “Preview” may be appropriate tasks/verbs associated with an item. The
15 shell may be furthered configured to launch applications in support of these tasks
16 upon a user selection to perform the action with respect to the item.

17 An item decoration view is sufficient to fully present a given item or a
18 homogeneous set of items, comprised of items having like item decoration views.
19 To display items with different item decoration schemas, the shell provides shell
20 view schemas that present items according to shell decoration views. A shell view
21 schema allows the shell or ISVs to declare appropriate views for given sets of
22 heterogeneous data.

23 Items chosen for representation within a shell decoration view may include
24 a common characteristic. A wide variety of common characteristics may be
25 acceptable for a shell decoration view. For example, a shell view schema could

1 define a “Picture” view used to display common and appropriate fields and
2 metadata for all known picture types (e.g., .GIF, .JPEG, .BMP, .TIFF, etc). The
3 shell view schema overrides conflicting display attributes for a given item
4 decoration view and presents each picture item according the shell view schema.
5 As another example, the shell could provide a “Document” shell view that is
6 optimized around appropriate columns and metadata for the items produced by
7 typical productivity applications, such as word processing documents,
8 spreadsheets, or databases, even though the item decorations for each of these
9 items may vary greatly from each other. Such a view has value by providing
10 common properties among each of these documents. When later document types
11 are installed, the shell view will be able to present these new items according to
12 the consistent shell view even though the new type may not have been considered
13 when the view was first created.

14 The shell view schemas may provide a wide variety of display attributes
15 and ISVs may want to provide such shell views. The display attributes may
16 include, without limitation: the size of the preview pane, metadata to display
17 within the preview pane, custom controls to be used, and tasks and verbs
18 appropriate for the presented items.

19 The shell can present items according to an explorer display view. An
20 “explorer” may be referred to as a storage application and may be provided by the
21 shell or by ISVs. The explorer, for example, may enable a user to view, query,
22 navigate, launch into tasks, or organize selected items in a data store. The term
23 “explorer” should not imply a location where the displayed items reside, and terms
24 such as “activity center,” “viewer” and “library” may be used interchangeably
25 with “explorer” to describe a storage application.

1 An example explorer schema hierarchy includes a bottom layer which is an
2 item view schema. The item view schema provides the basic display needed to
3 represent an item, and the explorer view schema can defer or fall back upon its
4 display elements when required.

5 The explorer view schema also includes a shell view schema and explorer
6 decorations. The explorer decorations decorate the explorer as a whole and
7 provide display elements such as distinctive colors and branding elements. These
8 explorer decorations persist among the various views the explorer provides. A
9 wide variety of display attributes may be appropriate for the explorer decorations.
10 For example, data queries or tasks/verbs associated with the explorer items may be
11 appropriate for display with an explorer. Displayed tasks are oftentimes coupled
12 with an application capable of performing the task.

13 The explorer view schema may optionally include a shell view schema or
14 multiple shell view schemas. The shell view schema may be configured to
15 provide a shell view for a subset of explorer items. For example, an explorer may
16 be configured to display song items to a user. A first shell view schema may be
17 included to provide a display of albums and a second shell view schema may be
18 included to provide a display of song tracks. In this manner, both types of items
19 will have appropriate views within the explorer. As discussed above, the
20 utilization of shell view relates to the presentation of a set of items which,
21 optionally, may share a common characteristic.

22 The explorer may also rely on shell views included within the shell. If
23 items selected for presentation within an explorer are not supported by any of the
24 shell views included by the explorer, the shell may provide an appropriate shell
25 view for use within the explorer. Similarly and as discussed above, the explorer

1 may also fall back to an item display view or a default display view provided by
2 the shell. This functionality insures that any item which can be displayed by the
3 shell is also capable of display within the explorer. The explorer can be
4 configured to defer to these shell provided display schemas or may rely upon them
5 to, for example, provide a display for unanticipated data.

6 The Shell namespace 202 supports user interface functionality that allows
7 items to be manipulated and interacted with in a consistent manner regardless of
8 the location where the underlying data for the item (e.g., the file contents, the
9 image data, the contact information data, etc.) is stored. From the point of view of
10 an application designer, the functionality supported by Shell namespace 202
11 results in a better development experience at least in part because such items can
12 be treated in a consistent manner despite the underlying data being stored in
13 different locations. A coherent experience is provided to the designer when using
14 the Shell namespace 202 despite the potential presence of such different data
15 storage locations.

16 The Shell namespace 202 defines additional namespaces, including a
17 Controls namespace 302, a Dialogs namespace 304, an Addin namespace 306, and
18 a Desktop namespace 308. Each of these additional namespaces 302, 304, 306,
19 and 308 in the Shell namespace includes one or more services or components that
20 provide various functionality, such as controls, dialogs, and/or handlers, as
21 discussed in more detail below.

22 The Shell namespace 202 defines an object model that can be used by each
23 of the additional namespaces 302, 304, 306, and 308. This object model includes
24 the following objects:

25

- An ExplorerItem object (also referred to as an Item object) used to describe items that can be presented to the user by way of the Shell, such as a file folders, stacks, individual files, individual WinFS items (e.g., albums, songs, pictures, etc.), and so forth.
- A Libraries object used to describe a query against ExplorerItems.
- A ViewFields object (also referred to as a Properties object) used to project data from items (e.g., as represented by ExplorerItem objects) for display to the user. A projection defined between properties in the storage system and a ViewFields object can perform additional calculations or processing based on the ExplorerItems. For example, if a folder representing a storage device is being displayed and the designer wishes to include in that folder an amount of free space on that storage device, then the projection calculates the amount of free space based on the total capacity of the storage device and the sizes of the files (represented as ExplorerItems) stored on the storage device
- A StorageFavorites object used to describe a link to a dynamic list, which is a persistent form of a Library object. The StorageFavorites object is a combination of a query and a view for presenting the results of the query to the user.

This object model is discussed in further detail in U.S. Patent Application No. _____, Attorney Docket No. MFCP.109834 (MS# 306923.01), filed 10/23/03, entitled "System and method for presenting items to a user with a contextual presentation", which is hereby incorporated by reference.

1 Controls namespace 302 (“System.Windows.Explorer.Controls”) defines a
2 collection of re-usable user interface controls that can be used by applications. As
3 multiple different applications can make use of these controls, these controls can
4 be viewed as being re-usable. This collection of re-usable controls can simplify
5 design of the application because the application designers can rely on these
6 controls rather than being required to design their own versions of the controls.
7 Additionally, by using this collection of re-usable controls, a consistent look and
8 feel across multiple applications can be achieved.

9 The controls defined by Controls namespace 302 allow manipulation and/or
10 display of shells of the user interface, including manipulation and/or display of
11 items in the user interface. As discussed above, these items that can be displayed
12 within shells of the user interface represent any of a wide variety of kinds of data
13 and are not simply traditional “files” stored in a file system. Furthermore, the
14 controls defined by Controls namespace 302 allow application designers to use the
15 same controls when working with data from a wide variety of locations, resulting
16 in a consistent and coherent design experience.

17 The Controls namespace 302 defines the following re-usable controls:

- 18 • An ExplorerView control is used to encapsulate the storage user
19 experience. The storage user experience is defined to work across
20 items with relationships, queries, and so forth. This storage user
21 experience allows end users to, for example, issue queries based on
22 property values, stack WinFS items with common property values,
23 allow drag and drop from non-stacked items to stacks (e.g. by setting
24 the value of a property in the property predicate to what exists in the
25 stack), and so forth. The ExplorerView control can also be used to

1 describe a folder for display having the same appearance as can be
2 found in previous operating systems (also referred to as legacy
3 systems), such as previous versions of the Microsoft® Windows®
4 operating system (e.g., the Windows® XP or Windows® 98 versions
5 of the Windows® operating system).

- 6 • An ExplorerItemView control used to display ExplorerItems. The
7 layout and design of the ExplorerItems can be defined by the
8 application designer. The operating system may also define a user
9 experience that imposes some constraints on the layout and design
10 that can be defined by the application designer, so the layout and
11 design defined by the application designer is consistent with the user
12 experience defined by the operating system. Thus, the
13 ExplorerItemView control allows the designer to determine how the
14 data from the ExplorerItems is to be displayed to the user (e.g., in
15 columns, rows, or some other configuration) as well as what data is
16 to be displayed to the user.
- 17 • A BasketControl control used to allow items to be added to a sidebar
18 (sidebars are discussed in additional detail below). An item can be
19 added to the sidebar in a "drag and drop" manner by using a cursor
20 control device to select an item and drag it over the sidebar, where it
21 can be dropped (e.g., by releasing a cursor control button). The
22 BasketControl control is discussed in further detail in U.S. Patent
23 Application No. _____, Attorney Docket No. 003797.00695 (MS#
24 304631.1), filed 10/13/03, entitled "Extensible Creation And Editing

1 Of Integrated Collections", which is hereby incorporated by
2 reference.

3 A ListMaker control used to allow items to be added to a list or
4 group. Lists or groups of items can be created in a "drag and drop"
5 manner by using a cursor control device to select an item and drag it
6 over the appropriate list or group, where it can be dropped (e.g., by
7 releasing a cursor control button). For example the items may be
8 songs and the list a playlist, the items may be images and the list a
9 photo album, and so forth.

- 10 • A PreviewImageControl component used to allow preview images
11 of items to be presented to the user. When an item is selected, the
12 preview image can be presented to the user, the preview image
13 typically being larger than a thumbnail for the image but less than
14 the entire display area of the user interface.

15 Dialogs namespace 304 ("System.Windows.Explorer.Dialogs") defines a
16 collection of user interface dialogs and wizards that can be used by applications.
17 Similar to the controls in the Controls namespace 302, the dialogs and wizards in
18 the Dialogs namespace 304 can be re-used so that multiple different applications
19 can make use of these dialogs and wizards. This collection of re-usable dialogs
20 and wizards helps maintain a consistent look and feel across multiple applications.

21 The dialogs and wizards defined by Dialogs namespace 304 allow actions
22 to be performed within a shell of the user interface. As discussed above, these
23 shells can be based on items that can represent any of a wide variety of kinds of
24 data and are not simply traditional "files" stored in a file system. Furthermore, the
25 dialogs and wizards defined by Dialogs namespace 304 allow application

1 designers to use the same dialogs and wizards when working with data from a
2 wide variety of locations, resulting in a consistent and coherent design experience.

3 The Dialogs namespace 304 defines the following dialogs and wizards:

- 4 • Open/Save dialogs used to allow files and folders to be opened and
5 saved. These dialogs are also extensible, allowing application
6 designers to extend the functionality of the dialogs. For example,
7 these dialogs can be extended to make use of various additional meta
8 data regarding the files and/or folders that the application designer
9 desires to use.
- 10 • CD/DVD Burn wizards used to allow optical discs, such as CDs
11 and/or DVDs, to be written to. Hardware designers can provide their
12 own wizards including their own hardware-specific steps that allow
13 CDs and/or DVDs to be written to.
- 14 • Send Photos wizard used to assist users in sending images by
15 electronic mail. The wizard allows the user to re-size the image to a
16 particular size. The wizard may re-size the image to a particular size
17 deemed suitable for electronic mailing, or alternatively may allow
18 the user to select from two or more different sizes.

19 Addin namespace 306 (“System.Windows.Explorer.Addin”) defines a
20 collection of base classes and interfaces for extending the user interface
21 functionality. Adding namespace 306 allows application designers to add
22 managed code that extends the functionality of the user interface. This extension
23 allows, for example, the application designers to customize portions of the user
24 interface as they see fit.

1 The Addin namespace 306 allows for the extension of shells of the user
2 interface that are used to display items, which represent any of a wide variety of
3 kinds of data and are not simply traditional “files” stored in a file system.
4 Furthermore, the Addin namespace 306 allows application designers to extend the
5 shells of the user interface in a coherent manner, providing a more managed and
6 improved design experience.

7 Managed code for various functionality is supported by the Addin
8 namespace 306, such as:

- 9 • Context menu handlers. Context menus can be accessed, for
10 example, by right-clicking on an item being displayed. The context
11 menu handlers allow application designers to define their own
12 entries to be added to these context menus.
- 13 • Thumbnail extractors. Thumbnail extractors allow application
14 designers to develop their own thumbnails (e.g., for files, folders,
15 and/or other items being displayed) and identify those thumbnails to
16 the operating system for display to the user.
- 17 • Handlers for composing/decomposing projections from WinFS to
18 Explorer. These handlers allow application designers to perform
19 various calculations when displaying information regarding files
20 and/or folders to the user (e.g., allowing the amount of free space on
21 a storage device to be calculated).

22 Desktop namespace 308 (“System.Windows.Explorer.Desktop”) defines a
23 collection of functionality that allows application designers to extend the
24 functionality of the desktop displayed to users. The Desktop namespace may be
25 part of the Shell namespace 202 (e.g., “System.Windows.Explorer.Desktop”), or

1 alternatively may be part of the presentation subsystem namespace 200 but not
2 part of the Shell namespace 202 (e.g., “System.Windows/Desktop”). The Desktop
3 namespace includes elements involved in the sidebar and notifications. Including
4 the sidebar and notifications elements in the Desktop namespace provides the user
5 with a centralized control for the sidebar and notifications. For example, rules
6 regarding notifications from all applications can be set by the user once, rather
7 than requiring the user to navigate through accessing rule definitions for each
8 application that uses notifications.

9 In general, the sidebar provides dynamic communication access and
10 information awareness in an integrated interactive peripheral awareness display
11 within which specified communications contacts and informational elements are
12 dynamically tracked or received and provided to a user on an ongoing basis. In
13 certain embodiments, this capability is provided via at least one customizable
14 dynamic thumbnail displayed in one or more columns in a persistent display strip
15 along one edge of a conventional display device. Further, in additional
16 embodiments, the thumbnails are displayed on any one portion or portions of a
17 display, including the entire display. The embodiment wherein the entire display
18 is covered is particularly useful where the sidebar will be used on a device having
19 a relatively small display area, such as, for example, a handheld or palm top
20 computing device, a cell phone, or any other electronic device having a limited
21 display area.

22 Each of the customizable dynamic thumbnails represents, for example,
23 particular communications contacts (such as, particular individuals, businesses,
24 organizations, or other entities) or particular elements of information that a user
25 may be interested in. Such information elements include, for example, when

1 shared files or folders are modified, when information in a shared database or
2 workspace changes, email status, calendars, Internet web pages, weather
3 conditions, appointments, schedules, statistical information, stock quotes, traffic
4 information, or any other Internet or network accessible information that may be
5 of interest to a user.

6 The aforementioned dynamic thumbnails generally comprise a combination
7 of a “tile” (also referred to as a “ticket”) describing the contact or information of
8 interest and a specialized “viewer” for displaying whatever communications
9 contact or information is represented by the tile. Each tile and viewer can, for
10 example, be an item in the storage system that can be accessed by way of the user
11 interface functionality provided by shell namespace 202. The one or more
12 “services” are used to automatically interact with, track, or receive the current
13 state of the information and/or status of the communications contacts described by
14 each tile. The current state of the information and the status of the
15 communications contacts are then dynamically provided by hosting each tile in a
16 “container” residing within an interactive peripheral awareness interface for
17 graphically and/or textually displaying the items. The peripheral awareness
18 interface displays information and/or communications contacts in such a way as to
19 reduce any potential distraction or interruption to the user.

20 In general, a tile is represented by a data structure such as an XML data file.
21 Each tile includes instructions as to what information or communications contact
22 is to be represented by the tile as well as pointers to particular services that
23 represent any of a number of conventional means for accessing and/or interacting
24 with the information or communications contacts. These services are automatically
25 or manually selected from a predefined or user definable library of services. In

1 particular, the different services represent shared code or functions that provide
2 functionality for accessing, receiving, retrieving, and/or otherwise interacting with
3 any conventional information, source of information, or communications contact.
4 These services are shared in the sense that they are used either alone, or in
5 combination, and may be used simultaneously by one or more tiles.
6 Consequently, it should be noted that in certain embodiments multiple services are
7 used in combination for providing complex interactions with any conventional
8 information, source of information, or communications contact.

9 One example of a service is the functionality necessary for monitoring an
10 email folder by connecting to a conventional MAPI server. Another example of a
11 service is functionality for sending or receiving email messages. Related services
12 provide functionality for communicating with contacts or transferring information
13 via any number of conventional methods, such as, for example instant messaging
14 or peer-to-peer communications schemes. Another example of a service is
15 functionality to convert a text file from one language to another. A further
16 example of a service is functionality necessary for monitoring a database. Still
17 other examples of services include functionality for receiving or retrieving data
18 from a web site or a remote server. Clearly, any method for interacting with any
19 information, source of information, or communications contact can be
20 implemented as a shared service for use by one or more tiles.

21 Further, as noted above, each tile's instructions includes a pointer to one of
22 a number of specialized viewers having the capability to display whatever type of
23 information or communications contact is represented by the tile. In other words,
24 each tile represents a combination of the information or contact that a user desires
25 to keep track of along with a definition of how the user desires to view that

1 particular information or contact as well as the ability to use any of a number of
2 services for accessing and interacting with the information or contact. Such access
3 or interaction can be accomplished locally, or across local intranets, extranets,
4 wired or wireless networks, the Internet, etc. via any conventional
5 communications protocol.

6 The viewers graphically display the tile as a resizable thumbnail or icon-
7 sized window having the information or contact data retrieved via one or more of
8 the services. In particular, the viewer is capable of dynamically displaying a tile
9 having textual, audible, or graphical information, including still or live images, or
10 any combination of textual, audible, or graphical information. For example, one
11 viewer type is capable of displaying contact information, i.e. a "person tile",
12 another is capable of displaying specific email information, such as, for example,
13 number of messages received, or number of messages from a particular source,
14 another viewer is designed to interact with a database to provide a summary of
15 particular information from the database in the thumbnail. Further examples of
16 viewer types include viewers capable of displaying still images, video images, a
17 summary of communications status, the results of a database query, etc. Clearly,
18 any type of viewer can be designed to be associated with any corresponding type of
19 information to ensure that any possible information can be displayed.

20 The sidebar is discussed in further detail in U.S. Patent Application No.
21 10/063,296, entitled "A system and process for providing dynamic communication
22 access and information awareness", which is hereby incorporated by reference.

23 The sidebar functionality of namespace 308 allows application designers to
24 add tiles to the sidebar.

1 Notifications (e.g., audible or visual notifications) are part of a user context
2 system that, in accordance with certain embodiments, includes three elements that
3 are compared for a decision as to how to process a notification. The first element
4 is the user's context (as may be provided by the operating system and arbitrary
5 programs that have extended it). The second element is the user's rules and
6 preferences. The third element is the notification itself (which contains elements
7 such as data and properties that may match the user's rules).

8 The user context system operates by the operating system and other
9 programs declaring a user's contexts, after which the system brokers the user's
10 context and rules. Notifications are raised by other programs calling into the
11 system. The user's context, rules, and elements of the notification are compared
12 and then a determination is made as to what should be done with the notification.
13 Examples of various options for what may be done with the notification include
14 denying (e.g., if the notification is not allowed to draw or make noise, and the
15 notification is to never be seen by the user), deferring (e.g., the notification is held
16 until the user's context changes or the user's rules dictate that it is subsequently
17 appropriate to deliver), delivering (e.g., the notification is allowed to be delivered
18 in accordance with the user's context and rules), and routing (e.g., the user's rules
19 indicate that the notification should be handed off to another system, regardless of
20 whether the notification is also allowed to be delivered in the present system).

21 In general, the user may be in a state deemed "unavailable" in which case
22 the notification is either not delivered or held until the user becomes "available".
23 For instance, if the user is running a full screen application, that user may be
24 deemed unavailable. Or, the user may be "available" but in such a state that the
25 notification needs to be modified to be appropriate for the user. For instance, if

1 the user is listening to music or in a meeting, the user may have indicated that the
2 notifications should be delivered to the user's screen but that the sound they make
3 should be either quieter or not made at all.

4 As noted above, the user context can determine in part whether
5 notifications are shown on the user's screen. When a notification is shown, it may
6 be shown based on certain gradients within the user context. In other words, there
7 are different levels of invasiveness of the form of the drawn notification that may
8 be specified. For example, a normal notification is free to pop out into the client
9 area and briefly obscure a window. If the user is in a slightly restrictive context,
10 the notification may be free to show, but only in a less invasive manner, such as it
11 might not be allowed to draw on top of another window. As another example, in
12 one embodiment where the user is running a maximized application, the default
13 setting may be that this means that context is slightly restricted, and that the user
14 has clearly made a statement that they want this application to get the entire client
15 area. In this setting, a notification may still be allowed to draw, but may be made
16 to only appear within the sidebar. In other words, this type of reduced
17 invasiveness in the notification drawing form lessens the impact of the
18 notification, and overall lessens the cognitive load.

19 User context systems are discussed in further detail in U.S. Patent
20 Application No. 10/402,179, filed 3/26/03, entitled "System and Method Utilizing
21 Test Notifications", which is hereby incorporated by reference.

22 The notifications functionality of namespace 308 allows application
23 designers to create their own notifications and have them displayed on the desktop
24 to notify the user when something that may be of interest to the user has happened
25 (e.g., an email message has been received, an instant message has been received,

1 and so forth). The notification functionality of namespace 308 further provides a
2 central location for users to define their rules and preferences for notifications
3 from multiple applications, releasing users from the burden of setting the same
4 rules and preferences multiple times (once for each of the multiple applications).

5 Additionally, a Contacts namespace defines a collection of controls and
6 dialogs for contact information. This contact information can be stored as items in
7 the storage system. A “contact” generally refers to any person, group,
8 organization, business, household, or other type of identifiable entity. “Contact
9 information” generally refers to any information that corresponds to a contact and
10 that may be considered relevant for identifying, contacting, accessing,
11 corresponding or communicating with the contact. Contact information is used by
12 an application to perform a desired function, such as, for example, sending an
13 email, initiating a telephone call, accessing a website, initiating a gaming session,
14 performing a financial transaction, and so forth. Non-limiting examples of contact
15 information include names, aliases, telephone numbers, email addresses, home
16 addresses, instant messaging (IM) addresses, and web addresses. Contact
17 information can also refer to other types of information such as the status of a
18 contact. For example, information indicating a contact is currently online, or on a
19 telephone line may also be broadly considered as contact information.

20 The Contacts namespace can be part of the presentation subsystem
21 namespace 200 but not part of the Shell namespace 202 , such as,
22 “System.Windows.Contacts”, “System.Windows.Controls” (controls 310 of Fig.
23 3), or “System.Windows.Collaboration”. Alternatively, the Contacts namespace
24 could be part of the Shell namespace 202. The Contacts namespace expands the
25 functionality of the System.Windows namespace, but need not be implemented as

1 part of the Shell (that is, need not be part of the System.Windows.Explorer
2 namespace 302).

3 The Contacts namespace allows contact information to be entered once by
4 the user yet be accessed by multiple different applications. The user is thus
5 released from the burden of entering the same contact information multiple times
6 (once for each of the multiple applications).

7 The controls and dialogs in the Contacts namespace include:

- 8 ● A Contact Picker dialog used to allow users to select or pick a
9 contact from their stored contacts. For example, allows users to
10 select a contact to which an electronic mail message will be sent, or
11 a contact for Instant Messaging. The user can browse through a list,
12 for example, of all of the user's contacts stored as items in the
13 storage system and select one or more of those contacts. The
14 Contact Picker dialog is discussed in further detail in U.S. Patent
15 Application No. 10/324,746, filed 12/19/02, entitled "Contact
16 Picker", which is hereby incorporated by reference.
- 17 ● A Contact Textbox control used to allow users to enter contact
18 information such as a name in freeform (e.g., by typing the name in)
19 and, as the contact information is entered, the control resolves the
20 information to one or more contacts from the user's stored contacts.
21 The resolved information can be displayed, for example, in a drop-
22 down menu or may be automatically entered into a type-in line
23 where the user was entering the name in freeform.

1 A Contact control used to display contact information to users.

2 Editing or freeform entry of contact information is not supported by
3 this control.

5 **EXAMPLE NAMESPACE MEMBERS**

6 This section includes multiple tables describing the examples of members
7 that may be exposed by example namespaces (e.g., namespaces in presentation
8 subsystem namespace 200 of Fig. 3). These exposed members can include, for
9 example, classes, interfaces, enumerations, and delegates. It is to be appreciated
10 that the members described in these examples are only examples, and that
11 alternatively other members may be exposed by the namespaces.

12 It should be appreciated that in some of namespace descriptions below,
13 descriptions of certain classes, interfaces, enumerations and delegates are left
14 blank. More complete descriptions of these classes, interfaces, enumerations and
15 delegates can be found in the subject matter of the compact discs that store the
16 SDK referenced above.

17

18 **System.Windows.Explorer.Controls**

19 The following tables list examples of members exposed by the
20 System.Windows.Explorer.Controls namespace.

21 **Classes**

22 **DefaultCommandEventArgs**

Contains event data for the
DefaultCommandEvent event.

23 **ExplorerView**

Allows a user to view information
about a folder's contents.

24 **FolderSelectionChangedEventArgs**

Contains event data for the
FolderSelectionChangedEvent event.

1	<u>IncludeItemEventArgs</u>	Contains event data for the <u>IncludeItemEvent</u> event.
2	<u>ItemVerbModifyEventArgs</u>	Initializes a new instance of the <u>ItemVerbModifyEventArgs</u> class.
3	<u>NavigationCompleteEventArgs</u>	Contains event data for the <u>NavigationCompleteEvent</u> event.
4	<u>NavigationFailedEventArgs</u>	Contains event data for the <u>NavigationFailedEvent</u> event.
5	<u>NavigationPendingEventArgs</u>	Contains event data for the <u>NavigationPendingEvent</u> event.

Enumerations

9	<u>FolderViewFlags</u>	Indicates the view properties for the <u>ExplorerView</u> .
10	<u>FolderViewMode</u>	Indicates the view mode for the <u>ExplorerView</u> .

Delegates

12	<u>DefaultCommandEventHandler</u>	Represents a method that will handle the <u>DefaultCommandEvent</u> .
13	<u>FolderSelectionChangedEventHandler</u>	Represents a method that will handle the <u>FolderSelectionChangedEvent</u> .
14	<u>IncludeItemEventHandler</u>	Represents a method that will handle the <u>IncludeItemEvent</u> .
15	<u>ItemVerbModifyEventHandler</u>	Represents a method that will handle the <u>ItemVerbModifyEvent</u> .
16	<u>NavigationCompleteEventHandler</u>	Represents a method that will handle the <u>NavigationCompleteEvent</u> .
17	<u>NavigationFailedEventHandler</u>	Represents a method that will handle the <u>NavigationFailedEvent</u> .
18	<u>NavigationPendingEventHandler</u>	Represents a method that will handle the <u>NavigationPendingEvent</u> .

1 **System.Windows.Explorer.Dialogs**

2 The following tables list examples of members exposed by the
3 System.Windows.Explorer.Dialogs namespace.

4 **Classes**

5 ColorDialog

6 CommonDialog

7 DialogControlItemSelectedEventArgs

8 FileDialog

9 An abstract class that is used as the
10 parent class of OpenFileDialog and
11 SaveFileDialogBase.

12 FileDialogCheckButton

13 Represents a check box control that
14 can be placed on a FileDialog.

15 FileDialogComboBox

16 An abstract class that is used as the
17 parent class of
FileDialogComboBox,
FileDialogOpenDropDown,
FileDialogRadioButtonGroup, and
FileDialogToolbarMenu.

18 FileDialogControlBase

19 An abstract class that is used as the
20 parent class of
FileDialogCheckButton,
FileDialogEditBox,
FileDialogPushButton, and
FileDialogContainerControlBase.

21 FileDialogControlItem

22 FileDialogEditBox

23 Represents a text box control that
24 can be placed on a FileDialog.

25 FileDialogOpenDropDown

26 FileDialogPushButton

27 Represents a button control that can
28 be placed on a FileDialog.

29 FileDialogRadioButtonGroup

30 FileDialogToolbarMenu

31 FileOkEventArgs

32 FileType

33 OpenFileDialog

34 Allows a user to select one or more
35 file to open. This class cannot be

1 [SaveAsFileDialog](#)

inherited.

2 Enables a user to choose a location
3 at which to save a file, and to
4 specify a filename. This class
5 cannot be inherited.

6 [SaveFileDialog](#)

7 Allows a user to choose a location
8 at which to save a file. This class
9 cannot be inherited.

10 [SaveFileDialogBase](#)

11 An abstract class that is used as the
12 parent class of [SaveAsFileDialog](#)
13 and [SaveFileDialog](#).

8 **Enumerations**

9 [FileDialogLayout](#)

10 [TileAttributes](#)

12 **Delegates**

13 [DialogControlItemSelectedEventHandler](#)

14 [FileOkEventHandler](#)

16 **System.Windows/Desktop**

17 The following tables list examples of members exposed by the
18 System.Windows/Desktop namespace. This namespace contains elements
19 involved in the sidebar and notifications.

21 **Classes**

22 [AnalogClockPanel](#)

23 [Appbar](#)

24 [AreaButton](#)

25 [BaseComTile](#)

26 [BaseSidebarClockSettings](#)

	<u>BaseTile</u>	An abstract class used as the parent class of a custom sidebar tile implementation.
1		
2	<u>BasketControl</u>	
3	<u>CalendarElement</u>	
4	<u>CalendarImages</u>	
5	<u>ChildrenWontFitArgs</u>	
6	<u>ClockHacks</u>	
7	<u>ClockPanel</u>	
8	<u>DataSourceEventArgs</u>	Represents event data passed from a data source.
9	<u>DigitalDateTimeElement</u>	
10	<u>DragButton</u>	
11	<u>DragControlWindow</u>	
12	<u>DraggableButton</u>	
13	<u>DragWindow</u>	
14	<u>DropArgs</u>	
15	<u>DropEventArgs</u>	
16	<u>ExtraSpaceArgs</u>	
17	<u>FillAlphaPresenter</u>	
18	<u>FillImageResourcePresenter</u>	
19	<u>FillPanel</u>	
20	<u>FillPresenter</u>	
21	<u>Flyout</u>	
22	<u>FlyoutLinkClickEventArgs</u>	Represents event data passed to an <u>RMAActionEventHandler</u> in response to an event that originated from the optional link found at the bottom of a sidebar tile's flyout view.
23	<u>FlyoutPresenter</u>	
24	<u>FlyoutStuff</u>	
25	<u>FocusableButton</u>	
	<u>FocusWithinWorkaroundHelper</u>	
	<u>FolderContentsChangedEventArgs</u>	Represents event data passed to a <u>FolderContentsChangedEventHandler</u>

in response to a change in the folder's contents.

1	
2	<u>GlobalSetting</u>
3	<u>HackBorder</u>
4	<u>HackImage</u>
5	<u>ImageButton</u>
6	<u>ImageResource</u>
7	<u>ImageResourcePresenter</u>
8	<u>ItemControl</u>
9	<u>ItemToolbarControl</u>
10	<u>ListMakerControl</u>
11	<u>MenuStuff</u>
12	<u>NativeResource</u>
13	<u>NativeResourceHelper</u>
14	<u>NativeResources</u>
15	<u>NativeResourceTypeConverter</u>
16	<u>NormalButton</u>
17	<u>Notification</u>
18	
19	<u>NotificationArea</u>
20	<u>NotificationButton</u>
21	
22	<u>NotificationClickedEventArgs</u>
23	
24	<u>NotificationContext</u>
25	

Represents a message and its associated data that the system sends to the user in a piece of user interface (UI).

Defines a button that appears on a notification.

Contains data associated with a click event in a notification window.

Declares whether an application is currently in a particular state. This state is used as part of the definition of the user's context.

<u>OverflowableCollection</u>	
<u>OverflowableControlCollection</u>	
<u>OverflowPanel</u>	
<u>OverflowPresenter</u>	
<u>OverFlowWrapper</u>	
<u>PanelInfo</u>	
<u>ProgressBar</u>	
<u>QuickLaunchTile</u>	
<u>RMAEventArgs</u>	Represents event data generated by a Rich Minimized Application (RMA). Consumed by an <u>RMAEventHandler</u> .
<u>RMAData</u>	
<u>SetdataArrayEventArgs</u>	Represents event data generated by a data array and sent to a <u>SetdataArrayEventHandler</u> .
<u>Sidebar</u>	
<u>SidebarAlarmClock</u>	
<u>SideBarClock</u>	
<u>SidebarClockSettings</u>	
<u>SlideShowTile</u>	
<u>StackAlphaPresenter</u>	
<u>StackImageResourcePresenter</u>	
<u>StartButton</u>	
<u>SyncHelper</u>	
<u>SyncItemBar</u>	
<u>SyncTile</u>	
<u>Taskbar</u>	
<u>TaskChevron</u>	
<u>TaskGroup</u>	
<u>TaskItem</u>	
<u>TaskList</u>	
<u>TaskOverflow</u>	
<u>TaskOverflowableControlCollection</u>	

1 TaskPresenter
2 TestTile
3 TestTimeZone
4 TextElementFontInfo
5 Theme
6 ThemeHelper
7 ThemeHelperOld
8 ThemeImage
9 ThemeImageTypeConverter
10 ThemeResources
11 Tile
12 TileCollection
13 TileOverflow
14 TileThumb
15 TileThumbDotNet
16 Timer
17 TimerStuff
18 TimeZone
19 ViewStatusControl
20 WebHostEventArgs
21 WindowOrigin
22 WindowOriginOld
23 WindowStuff

Interfaces

ICOMDataSourceHandler Used to create and communicate with a data source implemented in unmanaged code.

IOverflow

IOverflowList

ISidebar

Provides access to the sidebar, which hosts individual tiles. The sidebar host is responsible for opening and closing flyouts, displaying shortcut menus, and responding to events that involve the individual tiles.

ISidebarAlarm

Enumerations

AlarmState

BasketFlags

DragWindowPos

InvokeFolderActionEnum Enumeration for deciding what to do when a user invokes an ItemControl for a folder. Used with InvokeFolderAction and ItemControlInvokeFolderAction.

RMAAction

Constants used with the Action property to define an event action.

SelectionModeEnum

Delegates

ChildrenWontFitHandler

DocumentCompleteEventHandler

DropEventHandler

DropHandler

DummyDelegateToSetupAppDomainProperly

ExtraSpaceHandler

FlyoutClosingEventHandler

FlyoutLinkClickEventHandler

Represents the method that handles FlyoutLinkClickEvent events.

FolderContentsChangedEventHandler

NavigateErrorEventHandler

NotificationClickedEventHandler

Represents the method that handles NotificationClickedEvent events

NotifyCompleteEventHandler

Represents the method that handles

	<u>RMAActionEventHandler</u>	Represents the method that handles <u>RMAActionEvent</u> events.
	<u>SetdataArrayEventHandler</u>	Represents the method that handles <u>SetdataArrayEvent</u> events.

System.Windows.Controls

Provides classes and interfaces used by shell components.

The following tables list examples of members exposed by the

System.Windows.Controls namespace.

Classes

<u>ContactControl</u>
<u>ContactPickerDialog</u>
<u>ContactPropertyRequest</u>
<u>ContactPropertyRequestCollection</u>
<u>ContactSelection</u>
<u>ContactSelectionCollection</u>
<u>ContactTextBox</u>
<u>ContactTextBoxSelectionChangedEventArgs</u>
<u>ContactTextBoxTextChangedEventArgs</u>
<u>ContactTextBoxTextResolvedEventArgs</u>
<u>IncludeContactEventArgs</u>
<u>IteratedEventArgs</u>
<u>Iterator</u>
<u>OKCancelApplyEventArgs</u>
<u>OKCancelApplyStrip</u>

1 **Enumerations**

2 ContactControlPropertyPosition

3 ContactPickerDialogLayout

4 ContactPropertyCategory

5 Contact.PropertyType

6 ContactType

7 OKCancelApplyType

8 **Delegates**

9 ContactTextBoxSelectionChangedEventHandler

10 ContactTextBoxTextChangedEventHandler

11 ContactTextBoxTextResolvedEventHandler

12 IncludeContactEventHandler

13 IteratedEventHandler

14 OKCancelApplyEventHandler

15 OpenedEventHandler

16 EXEMPLARY COMPUTING SYSTEM AND ENVIRONMENT

17 Fig. 4 illustrates an example of a suitable computing environment 400
18 within which the programming framework 132 may be implemented (either fully
19 or partially). The computing environment 400 may be utilized in the computer
20 and network architectures described herein.

21 The exemplary computing environment 400 is only one example of a
22 computing environment and is not intended to suggest any limitation as to the
23 scope of use or functionality of the computer and network architectures. Neither
24 should the computing environment 400 be interpreted as having any dependency

1 or requirement relating to any one or combination of components illustrated in the
2 exemplary computing environment 400.

3 The framework 132 may be implemented with numerous other general
4 purpose or special purpose computing system environments or configurations.
5 Examples of well known computing systems, environments, and/or configurations
6 that may be suitable for use include, but are not limited to, personal computers,
7 server computers, multiprocessor systems, microprocessor-based systems, network
8 PCs, minicomputers, mainframe computers, distributed computing environments
9 that include any of the above systems or devices, and so on. Compact or subset
10 versions of the framework may also be implemented in clients of limited
11 resources, such as cellular phones, personal digital assistants, handheld computers,
12 or other communication/computing devices.

13 The framework 132 may be described in the general context of computer-
14 executable instructions, such as program modules, being executed by one or more
15 computers or other devices. Generally, program modules include routines,
16 programs, objects, components, data structures, etc. that perform particular tasks
17 or implement particular abstract data types. The framework 132 may also be
18 practiced in distributed computing environments where tasks are performed by
19 remote processing devices that are linked through a communications network. In
20 a distributed computing environment, program modules may be located in both
21 local and remote computer storage media including memory storage devices.

22 The computing environment 400 includes a general-purpose computing
23 device in the form of a computer 402. The components of computer 402 can
24 include, by are not limited to, one or more processors or processing units 404, a

1 system memory 406, and a system bus 408 that couples various system
2 components including the processor 404 to the system memory 406.

3 The system bus 408 represents one or more of several possible types of bus
4 structures, including a memory bus or memory controller, a peripheral bus, an
5 accelerated graphics port, and a processor or local bus using any of a variety of
6 bus architectures. By way of example, such architectures can include an Industry
7 Standard Architecture (ISA) bus, a Micro Channel Architecture (MCA) bus, an
8 Enhanced ISA (EISA) bus, a Video Electronics Standards Association (VESA)
9 local bus, and a Peripheral Component Interconnects (PCI) bus also known as a
10 Mezzanine bus.

11 Computer 402 typically includes a variety of computer readable media.
12 Such media can be any available media that is accessible by computer 402 and
13 includes both volatile and non-volatile media, removable and non-removable
14 media.

15 The system memory 406 includes computer readable media in the form of
16 volatile memory, such as random access memory (RAM) 410, and/or non-volatile
17 memory, such as read only memory (ROM) 412. A basic input/output system
18 (BIOS) 414, containing the basic routines that help to transfer information
19 between elements within computer 402, such as during start-up, is stored in ROM
20 412. RAM 410 typically contains data and/or program modules that are
21 immediately accessible to and/or presently operated on by the processing unit 404.

22 Computer 402 may also include other removable/non-removable,
23 volatile/non-volatile computer storage media. By way of example, Fig. 4
24 illustrates a hard disk drive 416 for reading from and writing to a non-removable,
25 non-volatile magnetic media (not shown), a magnetic disk drive 418 for reading

1 from and writing to a removable, non-volatile magnetic disk 420 (e.g., a “floppy
2 disk”), and an optical disk drive 422 for reading from and/or writing to a
3 removable, non-volatile optical disk 424 such as a CD-ROM, DVD-ROM, or other
4 optical media. The hard disk drive 416, magnetic disk drive 418, and optical disk
5 drive 422 are each connected to the system bus 408 by one or more data media
6 interfaces 426. Alternatively, the hard disk drive 416, magnetic disk drive 418,
7 and optical disk drive 422 can be connected to the system bus 408 by one or more
8 interfaces (not shown).

9 The disk drives and their associated computer-readable media provide non-
10 volatile storage of computer readable instructions, data structures, program
11 modules, and other data for computer 402. Although the example illustrates a hard
12 disk 416, a removable magnetic disk 420, and a removable optical disk 424, it is to
13 be appreciated that other types of computer readable media which can store data
14 that is accessible by a computer, such as magnetic cassettes or other magnetic
15 storage devices, flash memory cards, CD-ROM, digital versatile disks (DVD) or
16 other optical storage, random access memories (RAM), read only memories
17 (ROM), electrically erasable programmable read-only memory (EEPROM), and
18 the like, can also be utilized to implement the exemplary computing system and
19 environment.

20 Any number of program modules can be stored on the hard disk 416,
21 magnetic disk 420, optical disk 424, ROM 412, and/or RAM 410, including by
22 way of example, an operating system 426, one or more application programs 428,
23 other program modules 430, and program data 432. Each of the operating system
24 426, one or more application programs 428, other program modules 430, and
25

1 program data 432 (or some combination thereof) may include elements of the
2 programming framework 132.

3 A user can enter commands and information into computer 402 via input
4 devices such as a keyboard 434 and a pointing device 436 (e.g., a “mouse”).
5 Other input devices 438 (not shown specifically) may include a microphone,
6 joystick, game pad, satellite dish, serial port, scanner, and/or the like. These and
7 other input devices are connected to the processing unit 404 via input/output
8 interfaces 440 that are coupled to the system bus 408, but may be connected by
9 other interface and bus structures, such as a parallel port, game port, or a universal
10 serial bus (USB).

11 A monitor 442 or other type of display device can also be connected to the
12 system bus 408 via an interface, such as a video adapter 444. In addition to the
13 monitor 442, other output peripheral devices can include components such as
14 speakers (not shown) and a printer 446 which can be connected to computer 402
15 via the input/output interfaces 440.

16 Computer 402 can operate in a networked environment using logical
17 connections to one or more remote computers, such as a remote computing device
18 448. By way of example, the remote computing device 448 can be a personal
19 computer, portable computer, a server, a router, a network computer, a peer device
20 or other common network node, and so on. The remote computing device 448 is
21 illustrated as a portable computer that can include many or all of the elements and
22 features described herein relative to computer 402.

23 Logical connections between computer 402 and the remote computer 448
24 are depicted as a local area network (LAN) 450 and a general wide area network
25

1 (WAN) 452. Such networking environments are commonplace in offices,
2 enterprise-wide computer networks, intranets, and the Internet.

3 When implemented in a LAN networking environment, the computer 402 is
4 connected to a local network 450 via a network interface or adapter 454. When
5 implemented in a WAN networking environment, the computer 402 typically
6 includes a modem 456 or other means for establishing communications over the
7 wide network 452. The modem 456, which can be internal or external to computer
8 402, can be connected to the system bus 408 via the input/output interfaces 440 or
9 other appropriate mechanisms. It is to be appreciated that the illustrated network
10 connections are exemplary and that other means of establishing communication
11 link(s) between the computers 402 and 448 can be employed.

12 In a networked environment, such as that illustrated with computing
13 environment 400, program modules depicted relative to the computer 402, or
14 portions thereof, may be stored in a remote memory storage device. By way of
15 example, remote application programs 458 reside on a memory device of remote
16 computer 448. For purposes of illustration, application programs and other
17 executable program components such as the operating system are illustrated herein
18 as discrete blocks, although it is recognized that such programs and components
19 reside at various times in different storage components of the computing device
20 402, and are executed by the data processor(s) of the computer.

21 An implementation of the framework 132, and particularly, the API 142 or
22 calls made to the API 142, may be stored on or transmitted across some form of
23 computer readable media. Computer readable media can be any available media
24 that can be accessed by a computer. By way of example, and not limitation,
25 computer readable media may comprise “computer storage media” and

1 “communications media.” “Computer storage media” include volatile and non-
2 volatile, removable and non-removable media implemented in any method or
3 technology for storage of information such as computer readable instructions, data
4 structures, program modules, or other data. Computer storage media includes, but
5 is not limited to, RAM, ROM, EEPROM, flash memory or other memory
6 technology, CD-ROM, digital versatile disks (DVD) or other optical storage,
7 magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage
8 devices, or any other medium which can be used to store the desired information
9 and which can be accessed by a computer.

10 “Communication media” typically embodies computer readable
11 instructions, data structures, program modules, or other data in a modulated data
12 signal, such as carrier wave or other transport mechanism. Communication media
13 also includes any information delivery media. The term “modulated data signal”
14 means a signal that has one or more of its characteristics set or changed in such a
15 manner as to encode information in the signal. By way of example, and not
16 limitation, communication media includes wired media such as a wired network or
17 direct-wired connection, and wireless media such as acoustic, RF, infrared, and
18 other wireless media. Combinations of any of the above are also included within
19 the scope of computer readable media.

20 Alternatively, portions of the framework may be implemented in hardware
21 or a combination of hardware, software, and/or firmware. For example, one or
22 more application specific integrated circuits (ASICs) or programmable logic
23 devices (PLDs) could be designed or programmed to implement one or more
24 portions of the framework.

25

1 A programming interface (or more simply, interface) may be viewed as any
2 mechanism, process, protocol for enabling one or more segment(s) of code to
3 communicate with or access the functionality provided by one or more other
4 segment(s) of code. Alternatively, a programming interface may be viewed as one
5 or more mechanism(s), method(s), function call(s), module(s), object(s), etc. of a
6 component of a system capable of communicative coupling to one or more
7 mechanism(s), method(s), function call(s), module(s), etc. of other component(s).
8 The term “segment of code” in the preceding sentence is intended to include one
9 or more instructions or lines of code, and includes, e.g., code modules, objects,
10 subroutines, functions, and so on, regardless of the terminology applied or whether
11 the code segments are separately compiled, or whether the code segments are
12 provided as source, intermediate, or object code, whether the code segments are
13 utilized in a runtime system or process, or whether they are located on the same or
14 different machines or distributed across multiple machines, or whether the
15 functionality represented by the segments of code are implemented wholly in
16 software, wholly in hardware, or a combination of hardware and software.

17 Notionally, a programming interface may be viewed generically, as shown
18 in Fig. 5 or Fig. 6. Fig. 5 illustrates an interface Interface1 as a conduit through
19 which first and second code segments communicate. Fig. 6 illustrates an interface
20 as comprising interface objects I1 and I2 (which may or may not be part of the
21 first and second code segments), which enable first and second code segments of a
22 system to communicate via medium M. In the view of Fig. 6, one may consider
23 interface objects I1 and I2 as separate interfaces of the same system and one may
24 also consider that objects I1 and I2 plus medium M comprise the interface.
25 Although Figs. 5 and 6 show bi-directional flow and interfaces on each side of the

1 flow, certain implementations may only have information flow in one direction (or
2 no information flow as described below) or may only have an interface object on
3 one side. By way of example, and not limitation, terms such as application
4 programming or program interface (API), entry point, method, function,
5 subroutine, remote procedure call, and component object model (COM) interface,
6 are encompassed within the definition of programming interface.

7 Aspects of such a programming interface may include the method whereby
8 the first code segment transmits information (where “information” is used in its
9 broadest sense and includes data, commands, requests, etc.) to the second code
10 segment; the method whereby the second code segment receives the information;
11 and the structure, sequence, syntax, organization, schema, timing and content of
12 the information. In this regard, the underlying transport medium itself may be
13 unimportant to the operation of the interface, whether the medium be wired or
14 wireless, or a combination of both, as long as the information is transported in the
15 manner defined by the interface. In certain situations, information may not be
16 passed in one or both directions in the conventional sense, as the information
17 transfer may be either via another mechanism (e.g. information placed in a buffer,
18 file, etc. separate from information flow between the code segments) or non-
19 existent, as when one code segment simply accesses functionality performed by a
20 second code segment. Any or all of these aspects may be important in a given
21 situation, e.g., depending on whether the code segments are part of a system in a
22 loosely coupled or tightly coupled configuration, and so this list should be
23 considered illustrative and non-limiting.

24 This notion of a programming interface is known to those skilled in the art
25 and is clear from the foregoing detailed description of the invention. There are,

1 however, other ways to implement a programming interface, and, unless expressly
2 excluded, these too are intended to be encompassed by the claims set forth at the
3 end of this specification. Such other ways may appear to be more sophisticated or
4 complex than the simplistic view of Figs. 5 and 6, but they nonetheless perform a
5 similar function to accomplish the same overall result. We will now briefly
6 describe some illustrative alternative implementations of a programming interface.

7

A. FACTORING

8 A communication from one code segment to another may be accomplished
9 indirectly by breaking the communication into multiple discrete communications.
10 This is depicted schematically in Figs. 7 and 8. As shown, some interfaces can be
11 described in terms of divisible sets of functionality. Thus, the interface
12 functionality of Figs. 5 and 6 may be factored to achieve the same result, just as
13 one may mathematically provide 24 , or $2 \times 2 \times 3 \times 2$. Accordingly, as
14 illustrated in Fig. 7, the function provided by interface Interface1 may be
15 subdivided to convert the communications of the interface into multiple interfaces
16 Interface1A, Interface 1B, Interface 1C, etc. while achieving the same result. As
17 illustrated in Fig. 8, the function provided by interface I1 may be subdivided into
18 multiple interfaces I1a, I1b, I1c, etc. while achieving the same result. Similarly,
19 interface I2 of the second code segment which receives information from the first
20 code segment may be factored into multiple interfaces I2a, I2b, I2c, etc. When
21 factoring, the number of interfaces included with the 1st code segment need not
22 match the number of interfaces included with the 2nd code segment. In either of the
23 cases of Figs. 7 and 8, the functional spirit of interfaces Interface1 and I1 remain
24 the same as with Figs. 5 and 6, respectively. The factoring of interfaces may also
25

1 follow associative, commutative, and other mathematical properties such that the
2 factoring may be difficult to recognize. For instance, ordering of operations may
3 be unimportant, and consequently, a function carried out by an interface may be
4 carried out well in advance of reaching the interface, by another piece of code or
5 interface, or performed by a separate component of the system. Moreover, one of
6 ordinary skill in the programming arts can appreciate that there are a variety of
7 ways of making different function calls that achieve the same result.

8

9 B. REDEFINITION

10 In some cases, it may be possible to ignore, add or redefine certain aspects
11 (e.g., parameters) of a programming interface while still accomplishing the
12 intended result. This is illustrated in Figs. 9 and 10. For example, assume interface
13 Interface1 of Fig. 5 includes a function call *Square(input, precision, output)*, a call
14 that includes three parameters, *input, precision and output*, and which is issued
15 from the 1st Code Segment to the 2nd Code Segment. If the middle parameter
16 *precision* is of no concern in a given scenario, as shown in Fig. 9, it could just as
17 well be ignored or even replaced with a *meaningless* (in this situation) parameter.
18 One may also add an *additional* parameter of no concern. In either event, the
19 functionality of square can be achieved, so long as output is returned after input is
20 squared by the second code segment. *Precision* may very well be a meaningful
21 parameter to some downstream or other portion of the computing system;
22 however, once it is recognized that *precision* is not necessary for the narrow
23 purpose of calculating the square, it may be replaced or ignored. For example,
24 instead of passing a valid *precision* value, a meaningless value such as a birth date
25 could be passed without adversely affecting the result. Similarly, as shown in Fig.

10, interface I1 is replaced by interface I1', redefined to ignore or add parameters
2 to the interface. Interface I2 may similarly be redefined as interface I2', redefined
3 to ignore unnecessary parameters, or parameters that may be processed elsewhere.
4 The point here is that in some cases a programming interface may include aspects,
5 such as parameters, that are not needed for some purpose, and so they may be
6 ignored or redefined, or processed elsewhere for other purposes.

C. INLINE CODING

It may also be feasible to merge some or all of the functionality of two separate code modules such that the “interface” between them changes form. For example, the functionality of Figs. 5 and 6 may be converted to the functionality of Figs. 11 and 12, respectively. In Fig. 11, the previous 1st and 2nd Code Segments of Fig. 5 are merged into a module containing both of them. In this case, the code segments may still be communicating with each other but the interface may be adapted to a form which is more suitable to the single module. Thus, for example, formal Call and Return statements may no longer be necessary, but similar processing or response(s) pursuant to interface Interface1 may still be in effect. Similarly, shown in Fig. 12, part (or all) of interface I2 from Fig. 6 may be written inline into interface I1 to form interface I1''. As illustrated, interface I2 is divided into I2a and I2b, and interface portion I2a has been coded in-line with interface I1 to form interface I1''. For a concrete example, consider that the interface I1 from Fig. 6 performs a function call square (*input, output*), which is received by interface I2, which after processing the value passed with *input* (to square it) by the second code segment, passes back the squared result with *output*. In such a

1 case, the processing performed by the second code segment (squaring *input*) can
2 be performed by the first code segment without a call to the interface.

3

4 D. DIVORCE

5 A communication from one code segment to another may be accomplished
6 indirectly by breaking the communication into multiple discrete communications.
7 This is depicted schematically in Figs. 13 and 14. As shown in Fig. 13, one or
8 more piece(s) of middleware (Divorce Interface(s), since they divorce
9 functionality and / or interface functions from the original interface) are provided
10 to convert the communications on the first interface, Interface1, to conform them
11 to a different interface, in this case interfaces Interface2A, Interface2B and
12 Interface2C. This might be done, e.g., where there is an installed base of
13 applications designed to communicate with, say, an operating system in
14 accordance with an Interface1 protocol, but then the operating system is changed
15 to use a different interface, in this case interfaces Interface2A, Interface2B and
16 Interface2C. The point is that the original interface used by the 2nd Code Segment
17 is changed such that it is no longer compatible with the interface used by the 1st
18 Code Segment, and so an intermediary is used to make the old and new interfaces
19 compatible. Similarly, as shown in Fig. 14, a third code segment can be introduced
20 with divorce interface DI1 to receive the communications from interface I1 and
21 with divorce interface DI2 to transmit the interface functionality to, for example,
22 interfaces I2a and I2b, redesigned to work with DI2, but to provide the same
23 functional result. Similarly, DI1 and DI2 may work together to translate the
24 functionality of interfaces I1 and I2 of Fig. 6 to a new operating system, while
25 providing the same or similar functional result.

1

E. REWRITING

2
3 Yet another possible variant is to dynamically rewrite the code to replace
4 the interface functionality with something else but which achieves the same
5 overall result. For example, there may be a system in which a code segment
6 presented in an intermediate language (e.g. Microsoft IL, Java ByteCode, etc.) is
7 provided to a Just-in-Time (JIT) compiler or interpreter in an execution
8 environment (such as that provided by the .Net framework, the Java runtime
9 environment, or other similar runtime type environments). The JIT compiler may
10 be written so as to dynamically convert the communications from the 1st Code
11 Segment to the 2nd Code Segment, i.e., to conform them to a different interface as
12 may be required by the 2nd Code Segment (either the original or a different 2nd
13 Code Segment). This is depicted in Figs. 15 and 16. As can be seen in Fig. 15, this
14 approach is similar to the Divorce scenario described above. It might be done, e.g.,
15 where an installed base of applications are designed to communicate with an
16 operating system in accordance with an Interface 1 protocol, but then the operating
17 system is changed to use a different interface. The JIT Compiler could be used to
18 conform the communications on the fly from the installed-base applications to the
19 new interface of the operating system. As depicted in Fig. 16, this approach of
20 dynamically rewriting the interface(s) may be applied to dynamically factor, or
21 otherwise alter the interface(s) as well.

22 It is also noted that the above-described scenarios for achieving the same or
23 similar result as an interface via alternative embodiments may also be combined in
24 various ways, serially and/or in parallel, or with other intervening code. Thus, the
25 alternative embodiments presented above are not mutually exclusive and may be

1 mixed, matched and combined to produce the same or equivalent scenarios to the
2 generic scenarios presented in Figs. 5 and 6. It is also noted that, as with most
3 programming constructs, there are other similar ways of achieving the same or
4 similar functionality of an interface which may not be described herein, but
5 nonetheless are represented by the spirit and scope of the invention, i.e., it is noted
6 that it is at least partly the functionality represented by, and the advantageous
7 results enabled by, an interface that underlie the value of an interface.

8

9 **Conclusion**

10 Although the invention has been described in language specific to structural
11 features and/or methodological acts, it is to be understood that the invention
12 defined in the appended claims is not necessarily limited to the specific features or
13 acts described. Rather, the specific features and acts are disclosed as exemplary
14 forms of implementing the claimed invention.

15

16

17

18

19

20

21

22

23

24

25